Bayesian Networks with R
An introduction

- Introduce to the essential concepts in conjunction with examples in the open-source statistical environment R.
- There are several packages on CRAN (The Comprehensive R Archive Network) dealing with Bayesian networks. They can be divided in two categories:
  - ✓ Those that focus only on manipulating the parameters of the network, on prediction and on inference, under the assumption that all variables are discrete, as **gRain**.
  - ✓ Those that deal with structure and parameter learning, as **bnlearn**, both for discrete and continuous variables.

Package gRain

gRain (Søren Højsgaard, 2010) is an R package for probability propagation in Bayesian Networks that does not implement any structure or parameter learning algorithm, so the Bayesian network must be completely specified by the user. Reference manual:

https://cran.r-project.org/web/packages/gRain/gRain.pdf

Install from CRAN Required packages to install gRain:

https://cran.r-project.org/web/packages/gRain/index.html

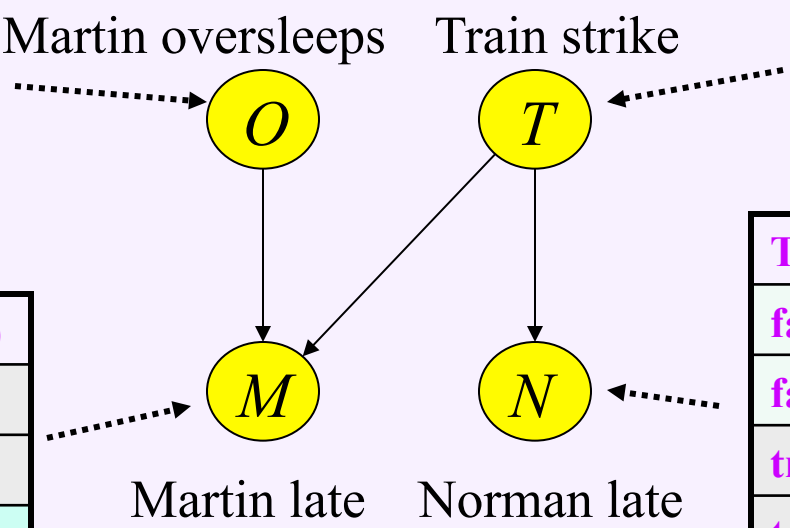
Required packages: gRbase, Rgraphviz and RBGL, obtained doing:

- source("http://bioconductor.org/biocLite.R")
- biocLite("RBGL”)
- biocLite("Rgraphviz")

Use: to perform exact inference with a BN specified by the user.

Let see how to use it going thru the example of Norman and Martin:

O	P(O)
false	0.6
true	0.4



T	P(T)
false	0.9
true	0.1

T	N	P(N/T)
false	false	0.9
false	true	0.1
true	false	0.2
true	true	0.8

T	O	M	P(M/T,O)
false	false	false	0.7
false	false	true	0.3
false	true	false	0.4
false	true	true	0.6
true	false	false	0.4
true	false	true	0.6
true	true	false	0.2
true	true	true	0.8

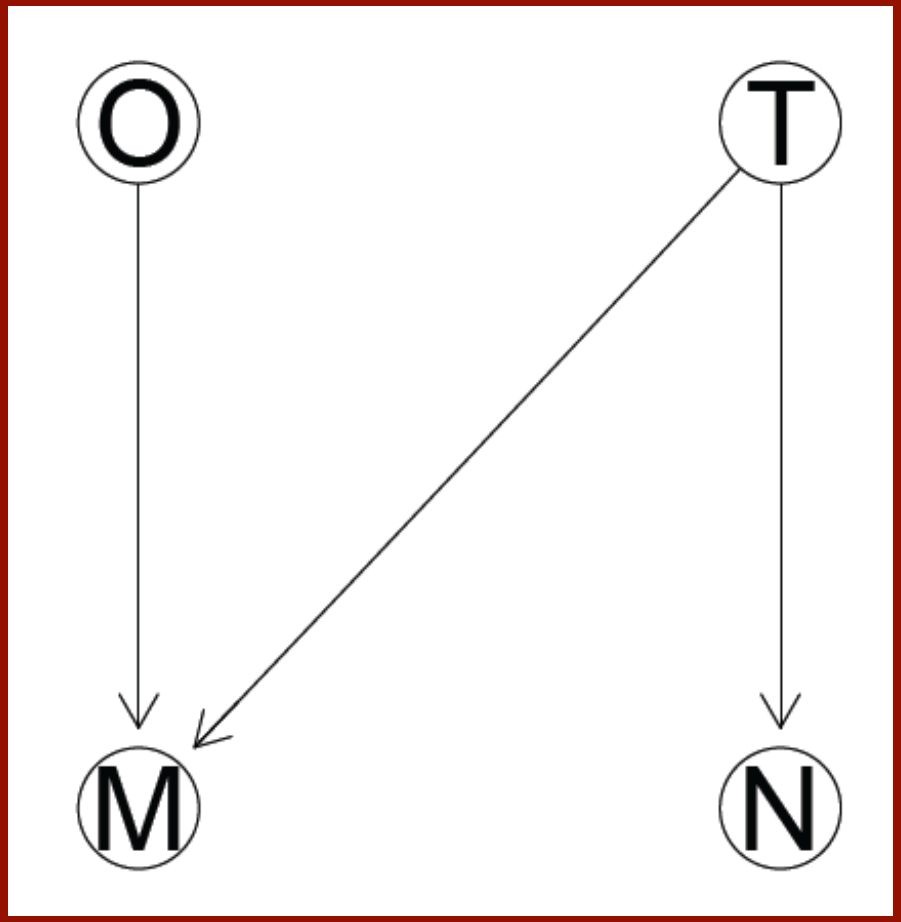
```
# First: which are the possible values of the nodes (all nodes are boolean):
tf<-c("false","true")
# Specify the CPTs:
node.O<-cptable(~ O, values=c(6,4),levels=tf)
node.T<-cptable(~ T, values=c(9,1), levels=tf)
node.N<-cptable(~ N + T, values=c(9,1,2,8), levels=tf)
node.M<-cptable(~ M + O + T, values=c(7,3,4,6,4,6,2,8), levels=tf)
# Create an intermediate representation of the CPTs:
plist<-compileCPT(list(node.O,node.T,node.N,node.M))
plist
plist$O
plist$T
plist$N
plist$M
```

```
> plist$O
O
false  true
  0.6   0.4
> plist$T
T
false  true
  0.9   0.1
> plist$N
       T
N       false true
  false   0.9  0.2
  true    0.1  0.8
```

```
> plist$M
, , T = false

       O
M       false true
  false   0.7  0.4
  true    0.3  0.6

, , T = true

       O
M       false true
  false   0.4  0.2
  true    0.6  0.8
```

Create a network of name ”Norman.net”, for instance:

```
Norman.net<-grain(plist)
summary(Norman.net)
# The graph:
plot1=plot(Norman.net)
plot1
```



We can compute the marginal probability
of each variable
These probabilities are EXACT!!

```
querygrain(Norman.net,nodes=c("O","T","N","M"),
type="marginal")
```

```
$O
O
false  true
  0.6   0.4

$T
T
false  true
  0.9   0.1

$M
M
false  true
0.554 0.446

$N
N
false  true
 0.83  0.17
```

We can also compute the joint probability of some nodes. For
instance:
#

```
querygrain(Norman.net,nodes=c("N","M"), type="joint")
```

#
that is, P(N=true,M=true)=0.0922,
P(N=true, M=false)=0.0778,
P(N=false, M=true)=0.3538,
P(N=false, M=false)=0.4762

```
        M
N         false   true
  false 0.4762 0.3538
  true  0.0778 0.0922
```

We can compute the probability of an event given an evidence.
If evidence is “N=true”, in order to compute the probability of the
other nodes, first we add the evidence to the network and name the
new BN Norman.net.2:

```
Norman.net.2<-setEvidence(Norman.net,nodes=c("N"),
states=c("true"))
```

The marginal distributions given
the evidence are:

```
marg=querygrain(Norman.net.2,nodes
=c("O","T","M"), type="marginal")
```

```
$O
O
false  true
  0.6   0.4

$T
T
    false      true
0.5294118 0.4705882

$M
M
    false      true
0.4576471 0.5423529
```

We can obtain the probability of the evidence used in Norman.net.2:

```
print(getEvidence(Norman.net.2))
```

```
Finding:
N: true
Pr(Finding)= 0.17
```

If the evidence now is: N=true & M=true, we construct a new
BN named Norman.net.3:
#

```
Norman.net.3<-setEvidence(Norman.net,nodes=c("N","M"),
states=c("true","true"))
```

#

```
# The marginals of the nodes O and T when the evidence is
# N=true & M=true, are:
#
nodos=c("O","T")
marg3=querygrain(Norman.net.3,nodes=nodos, type="marginal")
```

```
$O
O
    false      true
0.4880694 0.5119306

$T
T
    false      true
0.4099783 0.5900217
```

```
# The following gives the most probable value for each variable,
# given the evidences N=true & M=true.
# It does not imply that the jointly configuration with these values is
# the most probable!!
#
prediction=NULL
confidence.level=NULL
Node=NULL
for (i in 1:length(nodos))
   {
   prediction[i]<-tf[which.max(marg3[[i]])]
   confidence.level[i]<-max(marg3[[i]])
    }
   Node<-as.data.frame(cbind(nodos,prediction,confidence.level))
   Node
```

The following gives the most probable value for each variable,
given the evidences N=true & M=true.
It does not imply that the jointly configuration with these values is
the most probable!!
#

```
  nodos prediction  confidence.level
1     0       true 0.511930585683297
2     T       true  0.59002169197397
```

Instead, we can obtain the joint probability distribution of nodes O
and T given the evidences N=true & M=true, and see which is the
configuration which maximizes the probability:
#

```
predOT<-querygrain(Norman.net.3,nodes=c("O","T"), type="joint")
predOT
```

```
        T
O            false      true
  false 0.1757050 0.3123644
  true  0.2342733 0.2776573
```

```
which.max(predOT)
```

```
[1] 3
```

Therefore, the configuration that maximizes the joint probability
distribution of O and T, given the evidences N=true & M=true is:
O=false & T=true, with a confidence level of 0.3123644

Package bnlearn

bnlearn is an R package for Bayesian network structure learning (Block 4), parameter learning (Block 3) and approximate (not exact!) inference (Block 2).

Reference manual:

https://cran.r-project.org/web/packages/bnlearn/bnlearn.pdf

It can be downloaded from CRAN:

https://cran.r-project.org/web/packages/bnlearn/index.html

Graph package **Rgraphviz** can be installed from BioConductor:

- source("http://bioconductor.org/biocLite.R")
- biocLite("Rgraphviz")

Web site: **http://www.bnlearn.com**

Bnlearn (Marco Scutari, 2010, 2012) offers a wide variety of

- structure learning algorithms (Constraint-based algorithms, Score-based algorithms, and hybrid algorithms),
- parameter learning approaches (maximum likelihood for discrete and continuous data and Bayesian estimation for discrete data), and
- inference techniques.

It is the only package that keeps a clear separation between the structure of a network and the associated probability distribution, which are implemented as two different classes of R objects.

Uses:

- Structure learning.
- Parameter learning.
- Approximate inference with a learned BN .

Bayesian network structures with bnlearn

Ways of creating the graph structure of a BN (object of class bn):

1. Expert-driven approach: create a *custom Bayesian network* structure through three possible representations: the *arc set* of the graph, its *adjacency matrix* or a *model formula*.

   http://www.bnlearn.com/examples/dag

2. Data-driven approach: perform *Structure Learning* from data (Block 4), that is, use the data to determine which arcs are present in the graph that underlines the model.

   http://www.bnlearn.com/examples/score/

3. Hybrid approach: in some contexts we have prior knowledge on what the structure of the network should look like and we would like to incorporate such knowledge in the structure learning process. One way to do that is to use *whitelists* and *blacklists*.

   http://www.bnlearn.com/examples/whitelist/

Fitted Bayesian networks with bnlearn

Ways of creating a fitted BN (a bn object with parameters: bn.fit object):

1. Expert-driven approach: in which the parameters are specified by the user using custom.fit(), which takes a bn object encoding the network structure and a list with the parameters of the local distributions of the nodes.

   http://www.bnlearn.com/examples/custom/

2. Data-driven approach: learning it from a data set using bn.fit() and a network structure (bn object). (Block 3).

3. Hybrid approach: combining the above (using the assignment operator for bn.fit objects, which replaces the parameters of a single local distribution).

   http://www.bnlearn.com/examples/custom/